STATE OF ALABAMA

Information Technology Guideline

Guideline 660-01G1: Application Security - SQL Injection

1. INTRODUCTION:

No database is safe. An attack technique that has been widely used is Structured Query Language (SQL) Injection. SQL injection is a method for exploiting web applications that use client-supplied data in SQL queries. SQL Injection refers to the technique of inserting SQL meta-characters and commands into Web-based input fields in order to manipulate the execution of the back-end SQL queries. The parameters of a web-based application are modified in order to manipulate the SQL statements that are passed to the database to return data. For example, it is possible to cause a second query to be executed with the first by adding a single quote (‘) to the parameters.

SQL injection attacks can occur on Microsoft SQL Server, MySQL® and other SQL database servers, so no matter how strong the firewall rule sets are or how diligent the patching mechanisms may be, Web application developers must follow secure coding practices to prevent attackers from breaking into State database systems.

2. OBJECTIVE:

Know about the various types of SQL injection attacks that exist and plan ways to combat them on State of Alabama Web applications and SQL servers.

3. SCOPE:

These recommendations apply to administrators and developers (State of Alabama employees, contractors, vendors, and business partners) of State Web applications that access SQL database systems.

4. RECOMMENDATIONS:

Based on a best business practice of the Department of Defense (US Army issuance 08-EB-T-0002 dated 16 JAN 2008; titled Web Applications and SQL Injection) State of Alabama web application developers should apply the following recommendations to prevent SQL injection attacks.

SQL Injection happens when an application accepts user input that is directly placed into a SQL Statement and doesn't properly filter out dangerous characters. This can allow an attacker to not only steal data from the database, but also modify and delete it. Certain SQL Servers such as Microsoft SQL Server contain Stored and Extended Procedures (database server functions). If an attacker can obtain access to these procedures it may be possible to compromise the entire machine.

4.1 SQL INJECTION TECHNIQUES

SQL injection techniques include authorization bypass, using the SELECT command, using the INSERT command, and using SQL server stored procedures. Bypassing logon forms is the simplest SQL injection technique. Most web applications that use dynamic content of any kind will build pages using information returned from SELECT queries. SELECT queries are used to retrieve information from a database. The WHERE clause is the part of the query that is most susceptible to manipulation.

Attackers take advantage of the fact that programmers often chain together SQL commands with user-provided parameters, and can therefore embed SQL commands inside these parameters. The result is that the attacker can execute arbitrary SQL queries and/or commands on the backend database server through the Web application. The technique exploits web applications that use client supplied data in SQL queries.

Attackers commonly insert single quotes into a URL's query string, or into a forms input field to test for SQL Injection. If an attacker receives an error message, there is a good chance that the application is vulnerable to SQL Injection.

4.2 DETECTING SQL INJECTION

The most common way of detecting SQL injection attacks is by looking for SQL signatures in the incoming HTTP stream. For example, looking for SQL commands such as UNION, SELECT or xp_.

Pay attention to any occurrence of SQL specific meta-characters such as the single-quote, semicolon, or the double-dash (--). Be aware these signatures may result in a high number of false positives.

4.3 PROTECTION AGAINST SQL INJECTION ATTACKS

Limit user access. Use the principle of least privilege and ensure that the users created for the applications have the privileges needed and no more. All extra privileges, for example, PUBLIC ones, should not be available.

Delete unused accounts.

Do not enable the guest account.

Never use the default system account (sa). Always setup specific accounts for specific purposes.

Require authentication for all SQL Server accounts and ensure that passwords meet State standards.

Ensure that the mapping between database users and logins at the server level is correct. Run sp_change_users_login with the report option regularly to ensure that the mapping is as expected.

Secure the database and ensure that all excess privileges are removed.

Change database default passwords.

Remove all PUBLIC privileges where possible from the database.

Never grant permissions to the PUBLIC database role.

Run the listener as a non-privileged user.

Implement the design principle of least privilege on SQL servers. Run separate SQL Server services under separate accounts with the lowest possible privileges.

Reduce system exposure by running only the services and features that are necessary.

Remove or move to an isolated server the unused extended stored procedures, triggers, user-defined function, etc.

Remove sample databases from production servers.

Delete or archive installation files.

Remove culprit characters and character sequences. Safeguards against SQL injection include sanitizing the data and securing the application. To reduce the chance of an injection attack, remove characters and character sequences such as semicolons, double-dash (--), SELECT, INSERT, and xp_ from user input before building a query.

Escape quotes. The majority of injection attacks require the use of single quotes to terminate an expression. Do not allow dynamic SQL that uses concatenation, or at least filter the input values and check for special characters such as quote symbols. Using a simple function to convert all single quotes to two quotes reduces the chance of an injection attack succeeding.

Limit the length of user input. Keep all text boxes and form fields as short as possible to limit the number of characters that can be used to formulate an SQL injection attack. It is poor practice to have a text box on a form that can accept more characters in the field if the field you are comparing it against can only accept fewer characters.

Review the application source code. The solutions vary because the code can be written in many different languages (PHP, JSP, java, PL/SQL, VB, etc.). Review the source code for dynamic SQL where concatenation is used. Find the call that parses the SQL or executes it. Check back to where values are entered. Ensure that input values are validated and that quotes are matched and meta-characters are checked.

Minimize use of dynamic SQL. If dynamic SQL is necessary, use bind variables.

Minimize use of dynamic PL/SQL. If dynamic PL/SQL is necessary, use bind variables.

If PL/SQL is used, use AUTHID CURRENT_USER so that the PL/SQL runs as logged in user and not the creator of the procedure, function or package.

Restrict PL/SQL packages that can be accessed from apache

If concatenation is necessary then check the input for malicious code (i.e., check for UNION in the string passed in or meta-characters such as quotes) and use numeric values for the concatenation part.

Use a firewall (in accordance with State standards and guidelines).

Always block TCP port 1433 and UDP port 1434 on the perimeter firewall to include named instances that are listening on additional ports.

Never install SQL Server on a domain controller.

In a multi-tier environment, run Web logic and business logic on separate computers.

Do not change the default settings for the command xp_cmdshell. Do not grant execute permission on xp_cmdshell to users who are not members of the sysadmin role. Only members of the sysadmin role can execute xp_cmdshell.

Disable ad hoc data access on all providers except SQL OLE DB for all users except members of the sysadmin fixed server role.

Restrict membership of the sysadmin fixed server role to a trusted individual.

Set login auditing level to failure or all.

Enable security auditing of sysadmin actions, fixed role membership changes, all login related activity, and password changes.

Disable cross database ownership chaining. Use sp_dboption to enumerate and validate databases for which cross database ownership chaining is required and has been enabled.

Encrypt sensitive data so it cannot be viewed. Ensure encryption is compliant with state standards.

Install a certificate to enable SSL connections. Certificates should use the fully-qualified DNS name of the server.

Use the SQL Server service account to encrypt database files with EFS (Encrypting File System).

Verify the safety of stored procedures that have been marked for AutoStart.

Do not allow direct catalog updates.

Add Microsoft Baseline Security Analyzer (MBSA) to the weekly maintenance schedule, and follow up on any security recommendations that it makes.

5. DEFINITIONS:

SQL INJECTION: A type of security exploit in which the attacker adds Structured Query Language (SQL) code to a Web form input box to gain access to resources or make changes to data.

6. ADDITIONAL INFORMATION:

6.1 POLICY

Information Technology Policy 660-01: Application Security

6.2 RELATED DOCUMENTS

Information Technology Standard 620-03S1: Authentication-Passwords

Information Technology Standard 640-01S2: Secure Web App Deployment

Information Technology Guideline 660-02G2: Firewall Security

Information Technology Standard 680-03S1: Encryption

Signed by Art Bess, Assistant Director

7. DOCUMENT HISTORY

Version	Release Date	Comments
Original	3/21/2008	